INCORPORATION BY REFERENCE

The Java™ Language Specification (Sun Microsystems, Inc., 1993-95), (hereinafter referred to as the "Java language specification") a copy of which is attached hereto as Appendix A, incorporated herein by reference.

The Java Virtual Machine Specification (Sun Microsystems, Inc., 1993-95), (hereinafter referred to as the "Java virtual machine specification") a copy of which is attached hereto as Appendix B, incorporated herein by reference.

Ann Wollrath, et al., "A Distributed Object Model for Java™," an unpublished paper attached hereto as Appendix C, incorporated herein by reference.

b U. S. Patent Application Ser. No. 08/636,707, filed on even date herewith in the names of James H. Waldo, Krishna Bharat and Roger Riggs, and entitled " System And Method For Generating Identifiers For Uniquely Identifying Object Types For Objects Used In Processing Of Object-Oriented Programs And The Like" ~~(Atty. Docket No. P1091)~~ (hereinafter identified as the "Waldo, et al., patent application."), (now U.S. Patent No. 5,815,709) incorporated herein by reference.

FIELD OF THE INVENTION

The invention relates generally to the field of digital computer systems, and more particularly to systems and methods for facilitating the invocation by a program being processed by a computer in one address space, of processing of methods and procedures in another address space, which may be implemented either on the same computer or on another computer. The invention particularly provides a system and method for obtaining and dynamically loading "stub" information which facilitates invocation by a program operating in one address space of a remote method or procedure in another address space, and possibly on another computer.

BACKGROUND OF THE INVENTION

In modern "enterprise" computing, a number of personal computers, workstations, and other devices such as mass storage subsystems, network printers and interfaces to the public telephony system, are typically interconnected in one or more computer networks. The personal computers and workstations are used by individual users to perform processing in connection with data and programs that may be stored in the network mass storage subsystems. In such an arrangement, the personal computers/workstations, operating as clients, typically download the data and programs from the network mass storage subsystems for processing. In addition, the personal computers or workstations will enable processed data to be uploaded to the network mass storage subsystems for storage, to a network printer for printing, to the telephony interface for transmission over the public telephony system, or the like. In such an arrangement, the network mass storage subsystems, network printers and telephony interface operate as servers, since they are available to service requests from all of the clients in the network. By organizing the network in such a manner, the servers are readily available for use by all of the personal computers/workstations in the network. Such a network may be spread over a fairly wide area, with the personal computers/workstations being interconnected by communication links such as electrical wires or optic fibers.

In addition to downloading information from servers for processing, a client, while processing a program, can remotely initiate processing by a server computer of particular routines and procedures (generally "procedures"), in connection with certain "parameter" information provided by the client. After the server has processed the procedure, it will provide results of its processing to the client, which the client may thereafter use in its processing operations. Typically in such "remote procedure calls" the program will make use of a local "stub" which, when called, transfers the request to the server which implements the particular procedure, receives the results and provides them to the program. Conventionally, the stub must be compiled with the program, in which case the information needed to call the remote procedure must be determined at compile time, rather than at the time the program is run. Since the stub available to the client's programs is static, it may be at

best the closest that can be determined should be provided for the program when it (the program) is compiled. Accordingly, errors and inefficiencies can develop due to mismatches between the stub that is provided to a program and the requirements of the remote procedure that is called when the program is run.

SUMMARY OF THE INVENTION

The invention provides a new and improved system and method for facilitating the obtaining and dynamic loading of a stub provided to enable a program operating in one address space to remotely invoke processing of a method or procedure in another address space, so that the stub can be loaded by the program when it is run and needed, rather than being statically determined when the program is compiled. Indeed, the stub that is loaded can be obtained from the resource providing the remote method or procedure, and so it (the stub) can exactly define the invocation requirements of the remote method or procedure. Since the stub can be located and dynamically loaded while the program is being run, rather that being statically determined when the program is compiled, run-time errors and inefficiencies which may result from mis-matches between the stub that is provided and the requirements of the remote method or procedure that is invoked can be minimized.

In brief summary, the invention provides a stub retrieval and loading subsystem for use in connection with a remote method invocation system. The stub retrieval and loading subsystem controls the retrieval and loading of a stub for a remote method, into an execution environment, to facilitate invocation of the remote method by a program executing in the execution environment. The stub retrieval subsystem includes a stub retriever for initiating a retrieval of the stub and stub loader for, when the stub is received by the stub retriever, loading the stub into the execution environment, thereby to make the stub available for use in remote invocation of the remote method. In one embodiment, the stub retrieval and loading subsystem effects the retrieval and loading for a program operating in one address space provided by one computer, of stub class instances to effect the remote invocation of methods which are provided by objects operating in another address space, which may

be provided by the same computer or a different computer. In that same embodiment, the stub retrieval and loading subsystem effects the retrieval and loading of a stub class instance when the remote object is referenced, although in other embodiments retrieval and loading may be effected when the remote method is invoked.

BRIEF DESCRIPTION OF THE DRAWINGS

This invention is pointed out with particularity in the appended claims. The above and further advantages of this invention may be better understood by referring to the following description taken in conjunction with the accompanying drawings, in which:

FIG. 1 is a function block diagram of a computer network including an arrangement constructed in accordance with the invention for facilitating the obtaining, dynamic loading and use of "stub" information to enable a program operating in one address space to invoke processing of a remote method or procedure in another address space;

FIGs. ~~2 and 3~~ 2A through 3B are flow charts depicting the operations performed by the arrangement depicted in FIG. 1, which is useful in understanding the invention, with ~~FIG. 2~~ FIGs. 2A through 2C depicting operations performed in connection with obtaining and dynamic loading of the stub information and ~~FIG. 3~~ FIGs. 3A and 3B depicting operations performed in connection with use of the stub information to invoke processing of the remote method or procedure.

DETAILED DESCRIPTION OF AN ILLUSTRATIVE EMBODIMENT

FIG. 1 is a schematic diagram of a computer network 10 including an arrangement for facilitating dynamic loading of "stub" information to enable a program operating in one address space to remotely invoke processing of a method or procedure in another address space. With reference to FIG. 1, computer network 10 includes a plurality of client computers 11(1) through 11(N) (generally identified by reference numeral 11(n)), a plurality of server computers 12(1) through 12(M)

(generally identified by reference numeral 12(m)), all of which are interconnected by a network represented by a communication link 14. In addition, the network 10 may include at least one nameserver computer 13, which may also be connected to communication link 14, whose purpose will be described below. As is conventional, at least some of the client computers 11(n) are in the form of personal computers or computer workstations, each of which typically includes a system unit, a video display unit and operator input devices such as a keyboard and mouse (all of which are not separately shown). The server computers 12(m) and nameserver computer 13 also typically include a system unit (also not separately shown), and may also include a video display unit and operator input devices.

The client computers 11(n), server computers 12(m) and nameserver computer 13 are all of the conventional stored-program computer architecture. A system unit generally includes processing, memory, mass storage devices such as disk and/or tape storage elements and other elements (not separately shown), including network interface devices 15(n), 16(m) for interfacing the respective computer to the communication link 14. The video display unit permits the computer to display processed data and processing status to the operator, and an operator input device enables the operator to input data and control processing by the computer. The computers 11(n) and 12(m) and 13 transfer information, in the form of messages, through their respective network interface devices 15(n), 16(m) among each other over the communication link 14.

In one embodiment, the network 10 is organized in a "client-server" configuration, in which one or more computers, shown in FIG. 1 as computers 12(m), operate as servers, and the other computers, shown in FIG. 1 as computers 11(n) operate as clients. In one aspect, one or more of the server computers 12(m) may, as "file servers," include large-capacity mass storage devices which can store copies of programs and data which are available for retrieval by the client computers over the communication link 13 for use in their processing operations. From time to time, a client computer 11(n) may also store data on the server computer 12, which may be later retrieved by it (the client computer that stored the data) or other client computers for use in their processing operations. In addition, one or more of the server computers 12(m) may, as "compute servers," perform certain

processing operations in response to a remote request therefor from a client computer 11(n), and return the results of the processing to the requesting client computer 11(n) for use by them (that is, the requesting client computers 11(n)) in their subsequent processing. In either case, the server computers may be generally similar to the client computers 11(n), including a system unit, video display unit and operator input devices and may be usable by an operator for data processing operations in a manner similar to a client computer. Alternatively, at least some of the server computers may include only processing, memory, mass storage and network interface elements for receiving and processing retrieval, storage or remote processing requests from the client computers, and generating responses thereto. It will be appreciated a client computer 11(n) may also perform operations described herein as being performed by a server computer 12(m), and similarly a server computer 12(m) may also perform operations described herein as being performed by a client computer 11(n).

The network represented by communication link 14 may comprise any of a number of types of networks over which client computers 11(n), server computers 12(m) and nameserver computers 13 may communicate, including, for example, local area networks (LANs) and wide area networks (WANs) which are typically maintained within individual enterprises, the public telephony system, the Internet, and other networks, which may transfer digital data among the various computers. The network may be implemented using any of a number of communication media, including, for example, wires, optical fibers, radio links, and/or other media for carrying signals representing information among the various computers depicted in FIG. 1. As noted above, each of the computers typically includes a network interface which connects the respective computer to the communications link 14 and allows it to transmit and receive information thereover.

The invention provides a system for facilitating the obtaining and dynamic loading of "stub" information to enable a program operating in one address space to invoke processing of a remote method or procedure in another address space, which may be located on the same computer as the invoking program or on a different computer. The invention will be described in connection with programs provided in the Java™ programming language, as described in the Java language

specification, which are processed in connection with an execution environment which is provided by a Java virtual machine. The Java virtual machine, in turn, is specified in the Java virtual machine specification. As described in the Java language specification, programs in the Java programming language define "classes" and "interfaces." Classes are used to define one or more methods or procedures, each of which may be invoked by reference to an interface. A class may be associated with and extend a "super-class," and in that regard will incorporate all of the interfaces and methods of the super-class, and may also include additional interfaces and/or methods. A class may also have one or more sub-classes (and thus will comprise a super-class of each of its sub-classes), with each sub-class incorporating and possibly extending their respective super-classes.

An interface provides a mechanism by which a set of methods may be declared. In that connection, an interface identifies each method that is declared by the interface by, for example, a name, identifies the data type(s) of argument(s) that are to be provided for the method, the data type(s) of return values that are to be returned by the method, and identifiers for exceptions which can be thrown during processing of the method. A class may indicate that it implements a particular interface, and in that connection will include the program code which will be used in processing all of the methods which are declared in the interface. In addition, different classes may indicate that they implement the same interface, and each will have program code which will be used in processing all of the methods which are declared in the interface, but the program code provided in each class to for use in processing the methods may differ from the program code provided in the other classes which is used in processing the same methods; thus, an interface provides a mechanism by which a set of methods can be declared without providing an indication of the procedure which will be used in processing any of the methods. An interface may be declared independently of the particular class which implements the method or methods which can be invoked using the interface. In that regard, a class that invokes the method and a class that actually implements the method will not need to share a common super-class.

During processing of a Java program, as described in the Java virtual machine specification, a client computer 11(n) provides an execution environment 20 for interpreting the Java program. The

Java virtual machine includes a class loader 21 that, under control of a control module 19, can dynamically link instances of classes, generally identified in FIG. 1 by reference numeral 22, into the running program's execution environment while the program is being executed. In that operation, the control module 19 effectively enables the class loader to retrieve uninstantiated classes, which generally identified by reference numeral 23, instantiate them and link them as class instances 22 into the execution environment's address space at the Java program's run time as the methods which the respective classes 23 implement are called. In addition, the class loader 21 can discard ones of the class instances 22 when they are not needed or to conserve memory. It will be appreciated that, if a class instance 22 has been discarded, it may be reloaded by the class loader 21 at a later point if it is then needed.

The invention provides an arrangement which facilitates the remote invocation, by a program executing in an execution environment 20 by a client computer 11(n), of methods implemented by classes on a server computer 12(m). In executing a method, the server computer 12(m) will also provide an execution environment 24 for processing, under control of a control module 28, the Java method. In that operation, the Java virtual machine which provides the execution environment 21 includes a class loader 25 (which may be similar to the class loader 21) that, under control of the control module 28, can dynamically link an instance of the class 26, to enable the method to be processed in the execution environment 24, and instances of other classes (also generally represented by reference numeral 26) which may be needed to process the remotely-invoked method. In that operation, the control module 28 effectively enables the class loader 25 to retrieve an uninstantiated class for the method to be invoked, from a plurality of uninstantiated classes which are generally identified by reference numeral 27, instantiate it (that is, the uninstantiated class which provides the method to be invoked) and link it as a class instance 26 into the execution environment. In addition, the class loader 25 can discard the class instances 26 when processing of the method has terminated. It will be appreciated that, if class instances 26 has been discarded, it may be reloaded by the class loader 25 at a later point if it is then needed.

The structure of nameserver computer 13, if provided, is generally similar to that of the server computer 12(m), and will not be separately described.

To facilitate remote invocation of a method, the control module 19 of the client computer's execution environment 21 makes use of one or more stub class instances generally identified by reference numeral 30 which are provided as part of the execution environment 21 in which the various class instances 22, including the class instance which is invoking the remote method, are being processed. Each stub class instance 30 is an instance of an uninstantiated stub class 31, which the server computer 12(m) may maintain for the various class instances 26 and uninstantiated classes 27 which the server computer 12(m) has "exported," that is, which the server computer 12(m) makes available to client computers 11(n) for use in remote invocation of methods provided thereby. An uninstantiated stub class 31 includes declarations for the complete set of interfaces for the particular remote uninstantiated class 27 which implements the remote method to be invoked, and also provides or invokes methods which facilitate accessing of the remote method(s) which are implemented by the remote class. The uninstantiated stub class 31, when it is instantiated and provided to the execution environment 20 of the client computer 11(n) as a stub class instance 30, effectively provides the information which is needed by the control module 19 of the execution environment 20 of the invoking Java program, so that, when a remote method that is implemented by its associated class is invoked by a Java program running in a particular execution environment, the remote method will be processed and the return value(s) provided to the invoking Java program. In one embodiment, the arrangement by which the stub class instance may be provided to the execution environment 20 is similar to that described in the aforementioned Waldo, et al., patent application.

In addition, the server computer 12(m) provides a skeleton 32, which identifies the particular classes and methods which have been exported by the server computer 12(m) and information as to how it (that is, the server computer 12(m)) may load the respective classes and initiate processing of the particular methods provided thereby.

When a class instance invokes a remote method maintained by a server computer 12(m), it will provide values for various parameters to the stub class instance 30 for the remote method, which

values the remote method will use in its processing. If the remote method is implemented on the same computer as the invoking Java program, when the invoking Java program invokes a remote method, the computer may establish an execution environment, similar to the execution environment 20, enable the execution environment's class loader to load and instantiate the class which implements the method as a class instance similar to class instances 22, and process the remote method using values of parameters which are provided by the invoking class instance in the remote invocation. After processing of the method has been completed, the execution environment in which the remote method has been processed will provide the results to the stub class instance 30 for the remote method that was invoked, which, in turn, will provide to the particular class instance 22 which invoked the remote method.

Similar operations will be performed if client computer 11(n) and server computer 12(m) are implemented on different physical computers. In that case, in response to a remote invocation, the client computer 11(n) that is processing the invoking class instance 22, under control of the control module 19 for the execution environment 10 for the invoking class instance 22, will use the appropriate stub class instance 30 to communicate over the network represented by the communication link 14 with the server computer 12(m) which implements the remote method to enable it (that is, the server computer 12(m)) to establish an execution environment 24 for the class which implements the remote method, and to use the class loader 25 to load an instance of the class as a class instance 26. In addition, the client computer 11(n), also using the appropriate stub class instance 30, will provide any required parameter values to the server computer 12(m) over the network 14. Thereafter, the server computer 12(m) will process the remote method using parameter values so provided, to generate result value(s) which are transferred over the network to the client computer 11(n), in particular to the appropriate stub class instance 30. The client computer 11(n) will, after it receives the result value(s) from the network, provide them to the invoking class instance 22 for its processing.

In any case, when the control module 19 of the client computer's execution environment 20 determines that a reference to the remote object has been received, if it determines that the stub class

instance 30 is not present when it receives the reference, it will attempt to obtain the stub class instance 30 from, for example, the server computer 12(m) which implements the remote method, and enable the stub class instance 30 to be dynamically loaded in the execution environment 20 for the invoking class instance 22. A reference to the remote object may be received, for example, either as a return value of another remote method invocation or as a parameter that is received during another remote method invocation. The stub class instance may be dynamically loaded into the execution environment in a manner similar to that used to load class instances 22 in the execution environment 22. The execution environment 20 is provided with a stub class loader 33 which, under control of the control module 19, will attempt to find and load the stub class instances 30 as required by the class instances 22 processed in the execution environment. The location of a particular server computer 12(m) that maintains the class that implements a method to be invoked remotely may be included in the call from the invoking class instance or may be made known to the stub class loader 33 through another mechanism (not shown) maintained by the client computer 11(n).

However, if the stub class loader 33 is not otherwise notified of which server computer 12(m) maintains the class which implements a method which may be invoked remotely, it may use the nameserver computer 13 to provide that identification. The identification may comprise any identifier which may be used to identify a server computer 12(m) or other resource which is available on the network 14 and to which the server computer 12(m) can respond. Illustrative identifiers include, for example, a network address which identifies the server computer and/or resource, or, if the network 14 is or includes the Internet, an identifier to, for example, a World Wide Web resource which may provide the identification or a "uniform resource locator" ("URL") which provides a uniform mechanism for identifying resources that are available over the Internet. The server computer 12(m) which implements the remote method, in response to a request from the client computer 11(n) will provide stub class instance 30 which the client computer 11(n) may load into the execution environment 21 to thereafter enable the remote invocation to be initiated.

As noted above, if the stub class loader 33 does not know which server computer 12(m) implements the remote method which may be invoked (and thus does not know which computer is

to provide the stub class code for the remote invocation), it may, under control of the control module 19, obtain the identification from the nameserver computer 13. In that operation, the stub class loader 33 may use a previously-provided default stub class which is provided for use in such cases. The default class stub, when used by the invoking Java program, enables the computer that is processing the invoking Java program to communicate with the nameserver computer 13 to obtain information which can be used in invoking the remote method. This operation is essentially the same as the invocation of a remote method to be processed by the nameserver computer 13, with the remote method including a parameter identifying the class and method to be remotely invoked, and enabling the nameserver computer 13 to provide the identification of a server computer 12(m) which can process the method to the requesting client computer 11(n) and other information which may be helpful in communicating with the server computer 12(m) and invoking the particular method. It will be appreciated that the nameserver computer 13 will maintain a table (not separately shown) of "exported" resources, that is, resources, such as classes and methods, that are available to client computers 11(n) connected to the network 14, and information, such as the identifications of the particular server computers 12(m) which provide those resources, which will be useful to the client computers 11(n) in making use of the exported resources.

It will be appreciated that the nameserver computer 13 may create and maintain the exported resource table in a number of ways that are known in the art. For example, the nameserver computer 13 may periodically broadcast requests for exported resource information over the network 14, to which the various server computers 12(m) which maintain exported resources may respond; in that case, the nameserver computer 13 may establish its exported resource table based on the responses from the server computers 12(m). Alternatively, each of the various server computers 12(m) which maintains an exported resource may periodically broadcast information as to the exported resources which it maintains, and the nameserver computer 13 can update its exported resource table based on the broadcasts from the server computer. In addition, the nameserver computer's exported resource table may be established by a system operator and may be fixed until he or she updates it.

In any case, the information provided by the nameserver computer 13 in response to a request initiated by the default stub would include such information as, for example, the identification of a computer 12(m) which can provide a class which implements the remote method to be invoked, particular information which the computer (that is, the computer which implements the remote method) will require to provide the required stub class code, and the like. After receiving the information from the nameserver computer 13, the computer 11(n) that is processing the invoking Java program may, under control of the control module 19, use the information communicate with the computer (that is, the computer which implements the remote method) to obtain the stub class, and may thereafter invoke the method as described above.

With this background, the operations performed by client computer 11(n), server computer 12(m) and, if necessary, nameserver 13 in connection with obtaining and dynamic loading of a stub class instance when a reference to a remote method is received will be described in connection with the flow chart depicted in ~~FIG. 2~~ FIGs. 2A through 2C. In addition, operations performed by the client computer 11(n) and server computer in connection with remote invocation of a method using the stub class instance will be described in connection with the flow chart depicted in ~~FIG. 3~~ FIGs. 3A and 3B. With reference initially to FIG. 2A, the execution environment control module 19 will, when it receives a reference to a remote method, will initially determine whether an appropriate stub class instance is present in the execution environment 20 to facilitate invocation of the remote method (step 100). If the control module 19 determines that such a stub class instance 30 for the remote method is present in the execution environment, it may continue other operations (step 101). However, if the control module 19 determines in step 101 that such a stub class instance is not present in the execution environment 20 for the remote method, the control module 19 will use the stub class loader 33 to attempt to locate and load a stub class instance 30 for the class to process the remote method. In that case, the control module 19 will initially determine whether the invocation from the class instance 22 included a resource locator to identify the server computer 12(m) or other resource which maintains the class for the method to be invoked, or whether it (that is, the control module 19) or the stub class loader 33 otherwise are provided with such a resource locator (step 102). If the control module 19 makes a positive determination in that step, it will sequence to step 103 to enable the stub class loader 33

to initiate communications with identified server computer 12(m) to obtain stub class instance for the class and method to be invoked (step 103). When the stub class loader 33 receives the stub class instance 30 from the server computer 12(m), it will load the stub class instance 30 into execution environment 20 for the class instance 21 which initiated the remote method invocation call in step 100 (step 104). After the stub class instance 30 for the referenced remote method has been loaded in the execution environment, the method can be invoked as will be described below in connection with FIGs. 3A and 3B ~~FIG. 3~~.

Returning to step 102, if the control module 19 determines that the invocation from the class instance 22 did not include a resource locator to identify the server computer 12(m) or other resource which maintains the class for the method to be invoked, and further that it (that is, the control module 19) or the stub class loader 33 is not otherwise provided with such a resource locator, a "class not found" exception may be indicated, at which point the control module 19 may call an exception handler. The exception handler may perform any of a number of recovery operations, including, for example, merely notifying the control module 19 that the remote method could not be located and allow it to determine subsequent operations.

Alternatively, the control module 19 may attempt to obtain a resource locator from the nameserver computer 13 or other resource provided by the network 14 (generally represented in FIG. 1 by the nameserver computer 13), using a call to, for example, a default stub class instance 30. The call to the default stub class instance 30 will include an identification of the class and method to be invoked and the name of the nameserver computer 13(m). Using the default stub class instance 30, the control module 19 will enable the computer 11(n) to initiate communications with nameserver computer 13 to obtain an identifier for a server computer 12(m) which maintains the class and method to be invoked (step 110). The communications from the default stub class instance 30 will essentially correspond to a remote method invocation, with the method enabling the nameserver computer to provide the identification for the server computer 12(m), if one exists associated with the class and method to be remotely invoked, or alternatively to provide an indication that no server computer 12(m) is identified as being associated with the class and method. During the communications in step

110, the default stub class interface 30 will provide, as a parameter value, the identification of class and method to be invoked.

In response to the communications from the default stub class instance 30, the nameserver computer 13 will process the request as a remote method (step 111), with the result information comprising the identification for the server computer 12(m), if one exists that is associated with the class and method to be remotely invoked, or alternatively an indication that no server computer 12(m) is identified as being associated with the class and method. After finishing the method, the nameserver computer 13 will initiate communications with the default stub class instance 30 to provide the result information to the default stub class instance 30 (step 112).

After receipt of the result information from the nameserver computer 13, the default stub class instance, under control of the control module 19, will pass result information to the stub class loader 33 (step 113). Thereafter, the stub class loader 33 determines whether the result information from the nameserver computer comprises the identification for the server computer 12(m) or an indication that no server computer 12(m) is identified as being associated with the class (step 114). If the stub class loader 33 determines that the result information comprises the identification for the server computer 12(m), it (that is, the stub class loader 33) will return to step 101 to initiate communication with the identified server computer 12(m) to obtain stub class instance for the class and method that may be invoked. On the other hand, if the stub class loader 33 determines in step 114 that the nameserver computer 13 had provided an indication that no server computer 12(m) is identified as being associated with the class and method that may be invoked, the "class not found" exception may be indicated (step 115) and an exception handler called as described above.

As noted above, the stub class instance 30 retrieved and loaded as described above in connection with ~~FIG. 2~~ FIGs. 2A through 2C may be used in remote invocation of the method. Operations performed by the client computer 11(n) in connection with remote invocation of the method will be described in connection with the flow chart in ~~FIG. 3~~ FIGs. 3A and 3B. As depicted in FIG. 3A, when a class instance 22 invokes a method, the control module 19 may initially verify that a stub class instance 30 is present in the execution environment for remote method to be invoked (step 120). If a positive determination is

made in step 120, the stub class instance 30 will be used for the remote invocation, and in the remote invocation will provide parameter values which are to be used in processing the remote method (step 121). Thereafter, the stub class instance 30 for the remote method that may be invoked will be used to initiate communications with the server computer 12(m) which maintains the class for the remote method (step 122), in the process, the passing parameter values which are to be used in processing the remote method will be passed. It will be appreciated that, if the server computer 12(m) which is to process the method is the same physical computer as the client computer 12(n) which is invoking the method, the communications can be among execution environments which are being processed within the physical computer. On the other hand, if the server computer 12(m) which is to process the method is a different physical computer from that of the client computer 12(n) which is invoking the method, the communications will be through the client computer's and server computer's respective network interfaces 15(n) and 16(m) and over the network 14.

In response to the communications from the stub class instance in step 122, the server computer 12(m), if necessary establishes an execution environment 24 for the class which maintains the method that may be invoked, and the uses the information provided by the skeleton 32 to create a class instance 26 for that class (step 123). Thereafter, the server computer 12(m), under control of the control module 28, will process the method in connection with parameter values that were provided by stub class instance 30 (step 124). After completing processing of the method, the server computer 12(m), also under control of the control module 28, will initiate communications with the client computer's stub class instance 30 to provide result information to the stub class instance (step 125). In a manner similar to that described above in connection with step 102, if the server computer 12(m) which processed the method is the same physical computer as the client computer 12(n) which invoked the method, the communications can be among execution environments 24 and 20 which are being processed within the physical computer. On the other hand, if the server computer 12(m) which processed the method is a different physical computer from that of the client computer 12(n) which is invoking the method, the communications will be through the server computer's and client computer's respective network interfaces 16(m) and 15(n) and over the network 14. After the stub class instance 30 receives the result information from the server computer, it may provide result

information to the class instance 22 which initiated the remote method invocation (step 126), and that class instance 22 can continue processing under control of the control module 19.

Returning to step 120, if the control module 19 determines in that step that it does not have a stub class instance 30 that is appropriate for the remote method that may be invoked, it may at that point call an exception handler (step 127) to perform selected error recovery operations.

The invention provides a number of advantages. In particular, it provides a new system and method for facilitating dynamic loading of a stub which enables a program that is operating in one execution environment to remotely invoke processing of a method in another execution environment, so that the stub can be loaded by the program when it is run and needed. In systems in which stubs are compiled with the program, and thus are statically determined when the program is compiled, they (the stubs) may implement subsets of the actual set of remote interfaces which are supported by the remote references that is received by the program, which can lead to errors and inefficiencies due to mismatches between the stub that is provided to a program and the requirements of the remote procedure that is called when the program is run. However, since, in the dynamic stub loading system and method, the stub that is loaded can be obtained from the particular resource which provides the remote method, it (the stub) can define the exact set of interfaces to be provided to the invoking program at run time, thereby obviating run-time incompatibilities which may result from mis-matches between the stub that is provided and the requirements of the remote method that is invoked.

It will be appreciated that a number of modifications may be made to the arrangement as described above. For example, although the execution environment 20 has been described as obtaining and loading stub class instances to facilitate invocation of remote methods when references to the remote methods are received, it will be appreciated that stub class instances may instead be obtained and loaded when the remote methods are initially invoked. Obtaining and loading of the stub class instance for a remote method when a reference thereto is received will have the advantages that (i) the stub class instance will be present in the execution environment when the remote method is actually invoked, and (ii) if the appropriate stub class instance can not be located, the program or an operator may be notified at an early time. On the other hand, obtaining and loading of the stub

class instance for a remote method when the method is to be invoked may result in a delay of the invocation until the correct stub class instance can be found, if the method is in fact not invoked even if a reference to it is received the stub class instance may not need to be located and loaded.

It will be appreciated that a system in accordance with the invention can be constructed in whole or in part from special purpose hardware or a general purpose computer system, or any combination thereof, any portion of which may be controlled by a suitable program. Any program may in whole or in part comprise part of or be stored on the system in a conventional manner, or it may in whole or in part be provided in to the system over a network or other mechanism for transferring information in a conventional manner. In addition, it will be appreciated that the system may be operated and/or otherwise controlled by means of information provided by an operator using operator input elements (not shown) which may be connected directly to the system or which may transfer the information to the system over a network or other mechanism for transferring information in a conventional manner.

The foregoing description has been limited to a specific embodiment of this invention. It will be apparent, however, that various variations and modifications may be made to the invention, with the attainment of some or all of the advantages of the invention. It is the object of the appended claims to cover these and such other variations and modifications as come within the true spirit and scope of the invention.

What is claimed as new and desired to be secured by Letters Patent of the United States is: